Capstone: Deep Learning in Interpreting Digits

Lu Xing

1. Introduction

Deep learning is a very hot topic these days. In 2015, DeepMind's AlphaGo system learned the game of Go so well to beat a human professional Go player, which is a demonstration of achieving one of the long-standing "grand challenges" of Artificial Intelligence (AI). Basically, deep learning (also known as deep structured learning, hierarchical learning or deep machine learning) is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using a deep graph with multiple processing layers, composed of multiple linear and non-linear transformations[1,2,3,4,5,6,7,8,9].

Various deep learning architectures such as deep neural networks, convolutional deep neural networks, deep belief networks and recurrent neural networks have been applied to fields like computer vision[10], automatic speech recognition[11], natural language processing[12], audio recognition[13] and bioinformatics[14] where they have been shown to produce state-of-the-art results on various tasks.

Even our everyday life is full of deep learning applications, including querying on the searching engines, images categorization, Siri on the iOS and so on. In this project, I focus on the image recognition and their usage in helping people store numbers.

In machine learning and cognitive science, artificial neural networks (ANNs) is a network inspired by biological neural networks (the central nervous systems of animals, in particular the brain) which are used to estimate or approximate functions that can depend on a large number of inputs that are generally unknown[15]. Figure 1 shows a simple model of ANN, which has one input layer, followed by one hidden layer and the output layer gives the output.

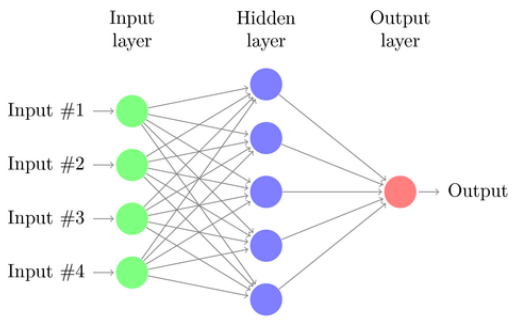

Figure 1: Simplified structure of ANN

The architecture of ANN resembles the network in the brain, which has a neuron, dendrite and synapse. In ANN, there are different kinds of neurons in different layers: input layer, hidden layer and output layer. There are connections between adjacent layers and different weights are applied to each connection. Besides architecture, ANN also needs activity rule and learning rule to specifying itself.

In this study, I apply deep learning to image recognition, mainly digit recognition. I trained neural networks to do the job. At first I tried pylearn2[16] but later turned to TensorFlow[17] which is more well-documented.

1.1 Probelm Statement

As more and more smartphones are in use, tons of images taken in everyday life are uploaded and shared by people. Also, Google's street view group is generating millions of images. Human beings could easily

understand and extract the meaning of each image, like the telephone number or street number. However, what if there are hundreds or thousands of images that are waiting for information extraction? So men have to teach machines to do that for us automatically. Numbers are very useful in maps, so in order to extract information by Google street view group, the first thing to do is to dig into how to interpret the digits in the images.

This is called Optical Character Recognition (OCR), which is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example from a television broadcast)[18]. There have been big breakthroughs in this area recently, partially because of the popularity of deep learning. There are also many great articles focusing on digit interpretation, like Goodfellow (2013a)[19], Goodfellow (2013b)[20] and Srivastava (2013)[21].

In this report, I will start with the simplest dataset: MNIST and move on to a more complexed dataset: cropped Street View House Numbers (SVHN) and full number SVHN. The detailed description of the datasets are in Sections 2 and 3.

1.2 Metrics

Then comes the question of what kind of machine learning algorithm is appropriate? As far as I know, different methods have been successfully operated on MNIST, including K-nearest neighbours, linear classifier, support vector machine, neural networks and more advanced convolutional neural nets. And I will use neural nets in this report.

The classification problem is defined as follows. Given dataset (x, y) with n examples, where x is a set of vectors with m features and y is a discrete label for each x. The goal is to produce from these examples a model $y = f(x)$ that will predict the classes y of future examples x with high accuracy[22]. The problem in this report falls in the scope of the classification problem. That is, for each digit, it only has ten discrete labels, which are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, while x is the matrix of each image.

The simplest metric for evaluating classification problem is accuracy: the number of correctly predicted points divided by the total points, as is indicated in *Equation (1)*.

$$Accuracy = \frac{\Sigma(label(predicted) == label(true))}{Count(Images)} \tag{1}$$

However, when the dataset is not balanced, this simple metric cannot be used. For example, if most of images are number 1, the model will just predict one all the time and it will also get a good accuracy, but it is not a robust model. So other metrics are still needed. Precision (also called positive predictive value) is the fraction of retrieved instances that are relevant, while recall (also known as sensitivity) is the fraction of relevant instances that are retrieved. The measures were defined in Perry, Kent and Berry (1955)[23]. *Equation (2)* and *Equation (3)* defines the calculation of precision and recall. *Equation (1)* can also be rewritten as *Equation (4)*.

$$Precision = \frac{Count(TP)}{Count(TP) + Count(FP)} \tag{2}$$

$$Recall = \frac{Count(TP)}{Count(TP) + Count(FN)} \tag{3}$$

$$Accuracy = \frac{Count(TP) + Count(TN)}{Count(P) + Count(N)} \tag{4}$$

- TP: True Positive

- FP: False Positive
- TN: True Negative
- FN: False Negative

Besides these, confustion matrix is also used in statisctical classification problem. Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class (or vice-versa)[24].

In sections 2, 3 and 4, if the model performance (prediction accuracy) is better than random guess, then this model is considered to be a practical model.

2 Warming up Dataset1: MNIST

2.1 Data Exploration and Visualization

The digit images in the MNIST set were originally selected and experimented with by Chris Burges and Corinna Cortes using bounding-box normalization and centering. Yann LeCun's version[25] uses centering by center of mass within in a larger window and I will use this version.The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The images have no color channel. The images are 28×28 and positioned at the center. Figure 2 shows some selected images from MNIST.

Figure 2: Selected images from the MNIST dataset

I count the number of different classes in the label set. There are 10 classes and they have similarly appearance frequency, as is shown in Figure 3.

2.2 Techniques

Since I will use ANN for the more complicated SVHN dataset, I first try ANN on this simpler dataset. TensorFlow is an open source software library for machine learning in various kinds of perceptual and language understanding tasks[26]. I can feed the images as matrix and the labels as vectors to the model. Images fall into different categories with different probabilities calculated by softmax regression. The digit with the highest probability is chosen to assign to the image. The network works as shown in Figure 4:

TensorFlow has a complete tutorial on how to use it for MNIST: https://www.tensorflow.org/versions/r0.10/tutorials/mnist/beginners/index.html#the-mnist-data. Also there is a course named "Deep Learning" available at Udacity.

Briefly, for one single image, the evidence for each digit is calculated by the weight and bias assigned to them. Then feed the evidence to the softmax regression (just like multi-variable logistic regression) and I have the probability for each digit.

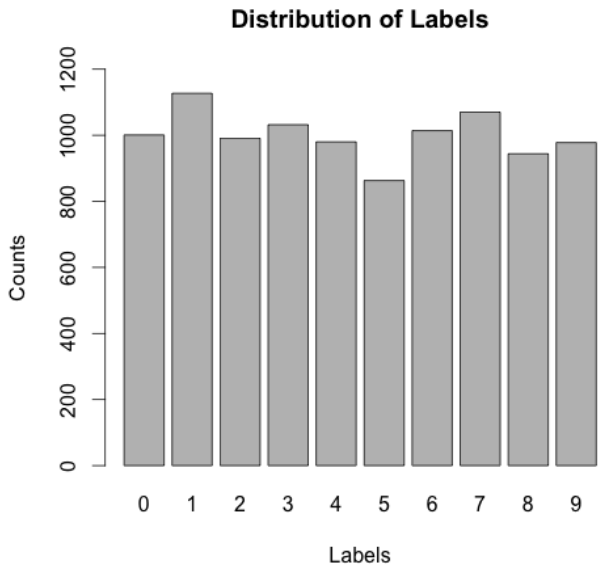

Figure 3: Distribution of Labels in MNIST

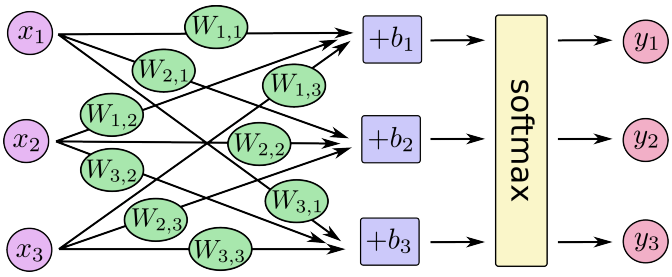

Figure 4: Softmax Regression for MNIST[27]

2.3 Data Preprocessing

MNIST is downloaded from: http://yann.lecun.com/exdb/mnist/.

Data needs to be preprocessed in order to be fed to the model described in Section 2.2.

Each image has a size of 28×28. To feed our training model with this, I need to flatten the images to one-dimension. The simplest way is to concatenate the rows sequentially. The corresponding label is the digit between 0 and 9. However, in libraries like tensorflow or pylearn2, the hot vector is required. So each label will become a vector with most elements being zero and the corresponding digit being one.

2.4 Implementation

I initialze the weight and bias to be zeros and define the softmax regression model. I monitor the cost using cross entropy and use gradient descent to find the minimized cross entropy with the learning rate initialized to 0.5. The training process run for 1000 times and each time it pick 100 images and labels randomly.

Please refer to `./warm_up1/tensorflow_mnist.py` for more details. In the terminal, change directory to `warm_up1`, and run `python tensorflow_mnist.py`.

2.5 Results

I use the data preprocessed in Section 2.3 and trained the model as indicated in Section 2.4 to predict the labels and compare them to the true labels. Random guess will have a prediction accuracy of 0.1. I evaluate the model based on the accuracy. Using this simplest model I get about 92%. This suggests that this model is actually 'learning' from the data and is a useful model.

2.6 Refinement and Discussion

As is suggested here: https://www.tensorflow.org/versions/r0.10/tutorials/mnist/pros/index.html#deep-mnist-for-experts, the single softmax regression has a bad performance. Using a multilayer convolutional network will improve the accuracy to 99%. I will come back to the convolutional network in the later part.

2.7 pylearn2 VS. TensorFlow

Pylearn2 is a Python library also capable for doing deep learning. At first, I used pylearn2. The images and labels are also preprocessed based on the procedure in Section 2.3. The batch size is set to 5 and the learning rate initialized at 0.05. Using this model I get testing data accuracy of 89.3%. The original output file is included in `./Additional_files/stdout_stderr/stdout_0816_1630.txt` and `./Additional_files/stdout_stderr/stderr_0816_1630.txt`. When I use the yaml file suggested in the 'http://nbviewer.jupyter.org/github/lisa-lab/pylearn2/blob/master/pylearn2/scripts/tutorials/softmax/_regression/softmax/_regression.ipynb' the best accuracy is 92.41%, better than tensorflow, however based on my learning experience using these two libraries, I would strongly suggest those inexperienced to use tensorflow because it does not require the knowledge of YAML file and the tutorial and the documents are all well structured.

So I would stick to tensorflow in the later parts.

In the terminal, change directory to `warm_up1`, and run `python pylearn2_create_dataset.py` first and use the generated `pkl` file to do the training. Run `python train.py pylearn2_train_mnist.yaml`. (If pylearn2 has been successfully installed and `pylearn2/scripts` already in the PATH environment variable then `python train.py pylearn2_train_mnist.yaml` will work fine, else `train.py` file is in the pylearn2 directory, which can be found at: https://github.com/lisa-lab/pylearn2/blob/master/pylearn2/scripts/train.py)

3 Warming up Dataset2: SVHN (Cropped Number)

3.1 Data Exploration and Visualization

Then I come to a more complexed dataset but with a simple image format. SVHN is a real-world image dataset[28], which can be downloaded at: http://ufldl.stanford.edu/housenumbers/. It is obtained from house numbers in Google Street View images. The SVHN data has over 600,000 digit images that come from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). There are two formats for downloading, one is original, variable-resolution, color house-number format with character level bounding boxes; the other one is MNIST-like format (e.g., the images are of small cropped digits).

I will use the MNIST-like format dataset in Section 3 and use the original `.png` format dataset in Section 4.

In this cropped-number dataset, all digits have already been resized to 32×32. Figure 5 shows some of the images in this cropped-number dataset.

First of all, like in Section 2, I look at the distribution over 10 different classes like in the SVHN dataset.

Figure 5: SVHN cropped numbers[28]

The labels here are not distributed evenly among all classes as is shown in Figure 6. As I mentioned in Section 1.2, I will not only use accuracy to evaluate our model, I will also use precision, recall and confusion matrix.

If random guess is applied, I will get an accuracy of 9.6%. So any model that predicts the labels with an accuracy higher than 9.6% will be a potential good model.

3.2 Techniques

I will build ANN for this dataset like in Section 2. Since this dataset is taken from the natural scene images, they are more complicated than the MNIST dataset. In Section 2, a multilayer convolutional network gave an accuracy of 99%. So I will also use a multilayer convolutional network. Then comes the question that I didn't answer in Section 2.6: what is convolutional network? For example, the images in the dataset are colorful, however, the digit in the images won't change because of the color of the images. Also, the digit won't change because of its position in the image. So when I feed the input data to the convolutional net, it will do something like filtering and then output a new format of data, as shown in Figure 7. It has several distinguished features: 3-dimension volumes of neurons; local connectivity; shared weights. I will implement this using tensorflow.

I have one convolutional layer followed by densely connected layer and softmax regression to the output.

1. First convolutional layer: convolution followed by max pooling. Patch size: 5×5. weight tensor: [5, 5, 1, 64].
2. Max pooling: 2×2.
3. Densely connected layer: image size 16×16, features: 64, channels: 1024
4. Dropout (reduce overfitting)
5. Softmax layer: output channels: 10

3.3 Data Preprocessing

Data needs to be preprocessed in order to be fed to the model described in Section 3.2.

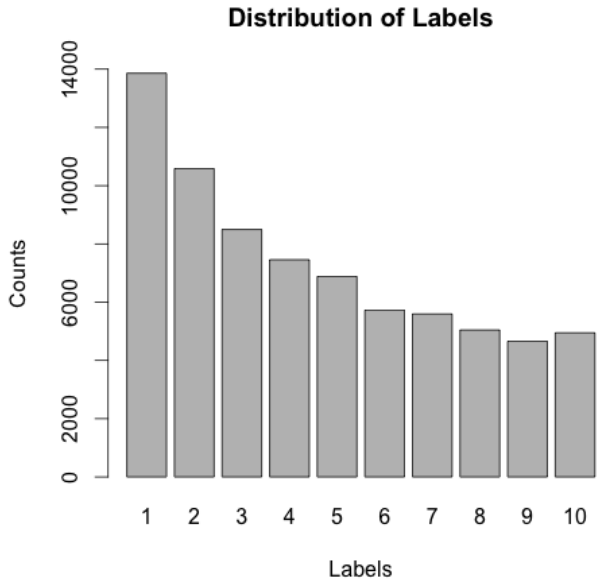

Figure 6: Distribution of Labels in SVHN (10 in the horizontal axes means label '0'

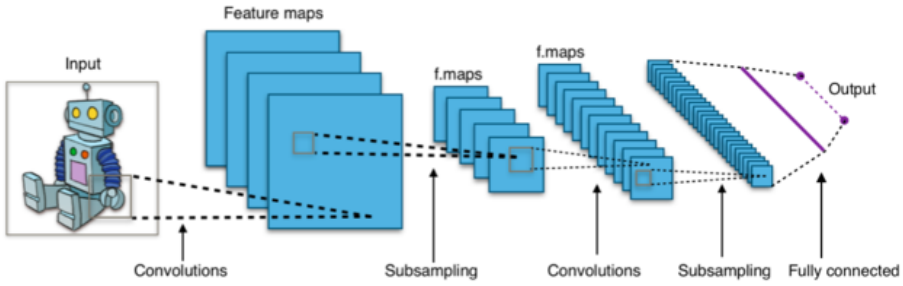

Figure 7: Convolution Neural Networks by Aphex34

First of all, I do data processing work. In order to reduce the size of the array and shorten the training time, I convert the color image to greyscale image. This preprocessing will not affect the training result of the images. Briefly, I worked in the following procedure:

1. Load `.mat` file and the images are transformed into numpy arrays.
2. Change the colored images to greyscale images. Reduce the three matrix (standing for three channels: R, G, B) to only one matrix.
3. Standardize the images: subtracting the mean and divided by standard deviation.

Then I convert the labels of the images to the hot vectors to fit the model.

3.4 Implementation

I use only one convolutional layer followed by the densely connected layer and softmat layer. I mainly follow the instructions at: https://www.tensorflow.org/versions/r0.10/tutorials/mnist/pros/index.html#deep-mnist-for-experts.

Due to the capacity of the personal computer, I could not feed the validation/testing data to the model all at once, which will cause the computer to collapse. Instead, I break the validation/testing data into small chunks (50 points a batch) and break the corresponding labels in the same way. Then I feed the data to the model and record result. I set up a numpy array to hold all the predictions. When all the data points are predicted, I will use `python scikit learn` to do calculations: `sklearn.metrics.precision_score`, `sklearn.metrics.recall_score`, `sklearn.metrics.confusion_matrix` and `sklearn.metrics.f1_score`.

3.5 Results

I use the data preprocessed in Section 3.3 and trained the model as indicated in Sections 3.2 and 3.4 to predict the labels and compare them to the true labels. I set the iteration number 10k as the stopping criteria and record the training accuracy and validation accuracy every ten bathches. Then I have plotted the accuracies of the training and validation dataset based on the training iterations, as shown in Figure 8.

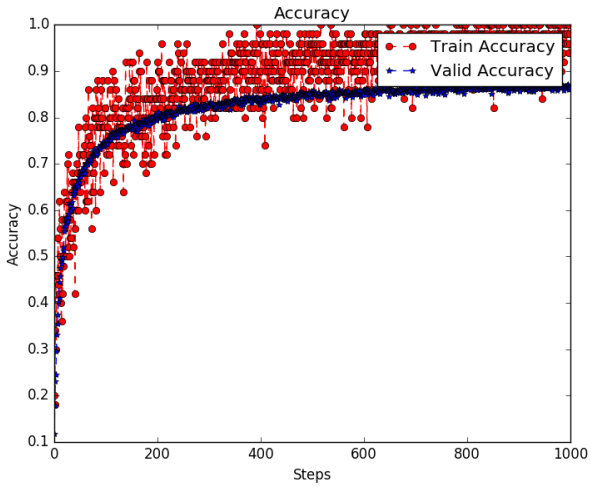

Figure 8: Accuracy of the training and validation dataset

Testing data precision = 0.849860389388
Testing data recall = 0.845843577136

The testing data accuracy is 0.845843577136 higher than the random guess, so this model is suitable for this task.

3.6 Refinement and Discussion

If one more convolutional layer and max-pooling is added, that is: the first weight tensor is [5,5,1,64], the second is [5,5,64,64]. Both of them are followed by one 2 $\times$ max-pooling layer and then a densely connected layer. I also set the iteration number as the stopping criteria but this time changes to 5k as shown in Figure 9.

Testing data precision = 0.849109945798
Testing data recall = 0.846996004917

From the above modification, I find that it is possible to increase the testing data recall/precision using one more convolutional layer and max-pooling layer, even with less iterations.

However, setting iteration number as the stopping criteria is not appropriate because I cannot stop the training when the model is already well trained and vice versa. As can be seen in Figure 8, the validation accuracy did not improve much during later iterations, so setting validation error as a stopping criteria is a

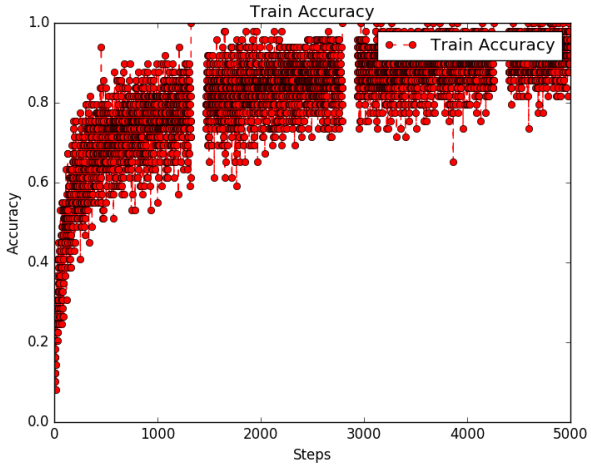

Figure 9: Accuracy of the training dataset

better choice. When the validation accuracy does not diminish (the absolute value between two adjacent validation accuracies are less than 1e-10), the training is stopped. The confusion matrix are included in the `./Additional_files/confusion_matrix/` directory. The results are included in the tables and the original output files are in the `./Additional_files/stdout_stderr` directory.

I use one layer and two layers separately.

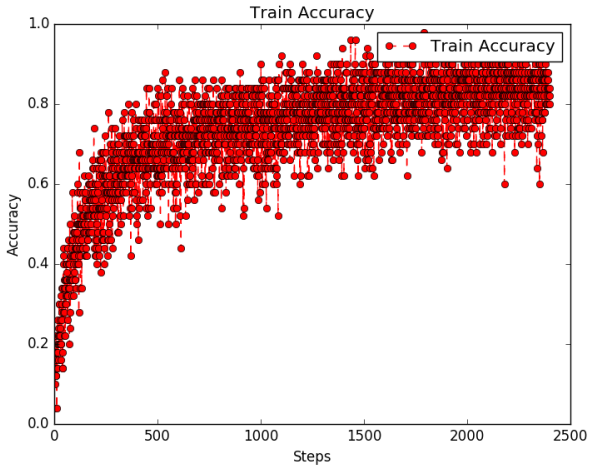

Figure 10: Accuracy of the training dataset

One layer: it took about 2400 iterations. The testing data accuracy is 0.78799170252. The training time is 35m39.553s. Confusion matrix is in `./Additional_files/confusion_matrix/confusion_1layer_cv0.8`. Figure 10 shows the training accuracy as the iteration increases.

Two layers: it took about 1500 iterations. The testing data accuracy is 0.809388444991. The training time is 22m49.214s. Confusion matrix is in `./Additional_files/confusion_matrix/confusion_2layers_cv0.8`. Figure 11 shows the training accuracy as the iteration increases.

Layer Number	Precision	Recall	f1_score
1	0.794124262489	0.78799170252	0.788897437526

Layer Number	Precision	Recall	f1_score
2	0.810027402529	0.809388444991	0.809118372852

In this dataset, the digit is positioned in the middle of the image, however in real images, it is seldom the case. So I could do distortions on the training data and see whether this distortions will help predict the correct classification. I refer to the tutorial at: https://github.com/tensorflow/tensorflow/blob/r0.10/tensorflow/models/image/cifar10/cifar10_input.py. The distortion methods include cropping the images, flipping, whitening, adjusting brightness and contrast. However, when I add this random distortion to the images and train the model again, I did not find any improvement on validation accuracy. The reason I think is that my validation set and testing set are not distorted images, so adding this distortion may not help improving accuracy, but it will help in predicting the real life images.

4 Dataset: SVHN (Full Number)

4.1 Data Exploration and Visualization

This dataset contains original images with character level bounding boxes. The bounding box information are stored in `digitStruct.mat`

Same as in Section 3, if random guess is applied, I will get an accuracy of 9.6%. So any model that predicts the labels with an accuracy higher than 9.6% will be a potential good model.

4.2 Techniques

I build 2-layer of CNN to predict. Each convolutional layer is folloId by the 2×2 max-pooling layer. Then it is followed the densely connected layer. Since the numbers are not single digits, I build five classifiers on top the network.

I have two convolutional layers followed by densely connected layer and five classifiers as Ill as softmax regression to the output.

1. First convolutional layer: convolution followed by max pooling. Patch size: 5×5. Iight tensor shape: [5, 5, 1, 64]. Bias tensor shape: [64].
2. Max pooling: 2×2.
3. Second convolutional layer: convolution followed by max pooling. Patch size: 5×5. Weight tensor shape: [5, 5, 64, 128]. Bias tensor shape: [128].
4. Max pooling: Max pooling: 2×2.
5. Densely connected layer: image size 8×8. Weight tensor shape: [$8 \times 8 \times 128$, 1024], channels: 1024
6. Dropout (reduce overfitting). keep_prob is 0.9 when training and is 1.0 when calculating the accuracy.
7. Build five classifiers on top of the network. Each has a weight tensor shape of [1024, 11] and bias tensor shape of [11].
8. Softmax layer on the five classifiers.

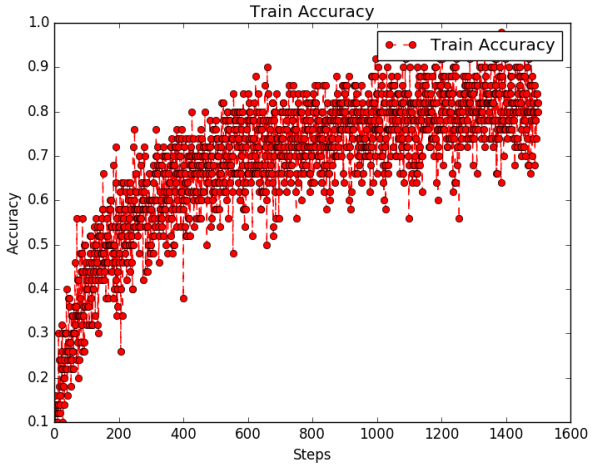

Figure 11: Accuracy of the training dataset

Figure 12: SVHN full numbers

4.3 Data Preprocessing

According to Goodfellow[19], I preprocessed our image data. The images are loaded and cropped based on the bounding boxes information. The bounding boxes are expanded by 30% in both x and the y direction[19]. The cropping process is adapted from hangyao[29]. After cropping, the images are resized to 32 $\times$ 32. I limit our data to learn from a 5 digit street number and discard the images with more than 5 digits.

4.4 Implementation

The testing data is split and is fed to the model 50 points at one time due to the hardware limitations (as mentioned in Section 3.4). I set the stopping criteria to be 60% accuracy for the validation set.

4.5 Results

I use the data preprocessed in Section 4.3 and trained the model as indicated in Sections 4.2 and 4.4 to predict the labels and compare them to the true labels. The testing data accuracy is 0.59435261708; precision

is 0.612978792357; recall is 0.59435261708; f1_score is 0.594232172676. The accuracy is higher than the random guess, so this model is suitable for this dataset.

4.6 Refinement and Discussion

The most straightforward parameter for ANN is number of the network layers. Here I use 1 layer, 2 layers and 3 layers convolutional neural nets and compare their performance using precision and recall. I also taken training time into consideration.

Layer Number	Precision	Recall	f1_score	Training Time (approximate)
1	0.578489442824	0.578489442824	0.561841404935	478min
2	0.611799366941	0.597260483624	0.595773606136	592min
3	0.611013925044	0.589531680441	0.591000835692	793min

I find that 3-layer and 2-layer are slightly better. Taking time into consideration, 2-layer is best.

However, in order to measure how robust the model is, k-fold cross validation is applied. Here I split the training data to 10 folds and record the validation accuracies as well as the standard deviation (Because of the huge size of the orginal training data and the age of my computer, I randomly pick half of the training data to do 10-fold cross validation to save the running time).

Layer Number	Validation Accuracy	Standard Deviation
1	0.599875	0.012762021461
2	0.613625	0.0217352077584
3	0.605875	0.0167483647364

As can be seen from this table, 1-layer model has a slightly lower standard deviation. However, the differences among the three models are not significant.

Next I use 1-layer model to do the comparison. I change the fixed learning rate to 10^{-1} and 10^{-7}. However, I find that 10^{-1} is too much for the model because the validation accuracy is oscillating and does not converge. The validation accuracy stays at about 3%. Also, 10^{-7} is too small. The validation accuracy stays at about 3% for a long time and converges slowly.

5. Conclusions

In Section 2, I used one layer network to do the work. Pylearn2 and tensorflow are both widely used libraries so I tried using both. They both have high testing data accuracies. However I used tensorflow in Sections 3 and 4, as pylearn2 is requiring the knowledge of YAML file and I am not very familiar with that.

In Section 3, I obtained a MNIST-format SVHN dataset and applied one-layer and two-layer CNN to it. As it turned out, the two-layer CNN did a better job with a higher precision and recall.

According to results in Section 4, I found that a two-layer CNN did best among the tested models. A three-layer CNN also had similar precision and recall to two-layer CNN, but it took a much longer time to train. However, there can still be improvements. First, the parameters can be tuned, including the tensor shape in the model, the learning rate, the cost function, the pooling processes and the number of the layers. Second, in order to do better with more real life images, preprocessing is needed. As is suggested by Goodfellow[19], new data can be obtained by modifying the existing data points, like distortion, whitening, translation to better train the model to learn from the real-world images. In this way, when new images are fed, the model will give better predictions. Third, usually I trained my models in several hours. In order to

do deep learning, the computer should be good enough and the memory should be big enough to hold the data. Or I can use GPU instead of CPU to do the work.

6. References

[1] Ian Goodfellow, Yoshua Bengio and Aaron Courville. Deep Learning. 2016. MIT Press. http://www.deeplearningbook.org/

[2] Li Deng and Dong Yu (2014), "Deep Learning: Methods and Applications", Foundations and Trends® in Signal Processing: Vol. 7: No. 3–4, pp 197-387. http://dx.doi.org/10.1561/2000000039

[3] Yoshua Bengio (2009), "Learning Deep Architectures for AI", Foundations and Trends® in Machine Learning: Vol. 2: No. 1, pp 1-127. http://dx.doi.org/10.1561/2200000006

[4] Bengio, Y.; Courville, A.; Vincent, P. (2013). "Representation Learning: A Review and New Perspectives". IEEE Transactions on Pattern Analysis and Machine Intelligence. 35 (8): 1798–1828. arXiv:1206.5538. doi:10.1109/tpami.2013.50

[5] Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". Neural Networks. 61: 85–117. arXiv:1404.7828. doi:10.1016/j.neunet.2014.09.003

[6] Bengio, Yoshua; LeCun, Yann; Hinton, Geoffrey (2015). "Deep Learning". Nature. 521: 436–444. doi:10.1038/nature14539

[7] Deep Machine Learning – A New Frontier in Artificial Intelligence Research – a survey paper by Itamar Arel, Derek C. Rose, and Thomas P. Karnowski. IEEE Computational Intelligence Magazine, 2013

[8] Schmidhuber, Jürgen (2015). "Deep Learning". Scholarpedia. 10 (11): 32832. doi:10.4249/scholarpedia.32832

[9] Carlos E. Perez. "A Pattern Language for Deep Learning"

[10] Bengio, Yoshua. "Learning deep architectures for AI." Foundations and trends® in Machine Learning 2.1 (2009): 1-127.

[11] Hinton, Geoffrey, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." IEEE Signal Processing Magazine 29.6 (2012): 82-97.

[12] Collobert, Ronan, and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning." Proceedings of the 25th international conference on Machine learning. ACM, 2008.

[13] Lee, Honglak, et al. "Unsupervised feature learning for audio classification using convolutional deep belief networks." Advances in neural information processing systems. 2009.

[14] Liu, Feng, et al. "De novo identification of replication-timing domains in the human genome by deep learning." Bioinformatics (2015): btv643.

[15] http://www.inference.phy.cam.ac.uk/itprnn/book.pdf

[16] Ian J. Goodfellow, et al. "Pylearn2: a machine learning research library". arXiv preprint arXiv:1308.4214

[17] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[18] https://dev.havenondemand.com/apis/ocrdocument#overview

[19] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. arXiv:1312.6082v4

[20] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, Yoshua Bengio. Maxout Networks. Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

[21] Srivastava, Nitish. Improving neural networks with dropout. Master's thesis, U. Toronto, 2013.

[22] Domingos, Pedro, and Geoff Hulten. "Mining high-speed data streams." Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2000.

[23] "Machine literature searching X. Machine language; factors underlying its design and development". 1955. doi:10.1002/asi.5090060411

[24] Powers, David Martin. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation." (2011).

[25] http://yann.lecun.com/exdb/mnist/

[26] "TensorFlow: Open source machine learning" "It is machine learning software being used for various kinds of perceptual and language understanding tasks" — Jeffrey Dean, minute 0:47 / 2:17 from Youtube clip

[27] https://www.tensorflow.org/versions/r0.10/tutorials/mnist/beginners/index.html#the-mnist-data

[28] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011.

[29] https://github.com/hangyao/street/_view/_house/_numbers/blob/master/3/_preprocess/_multi.ipynb